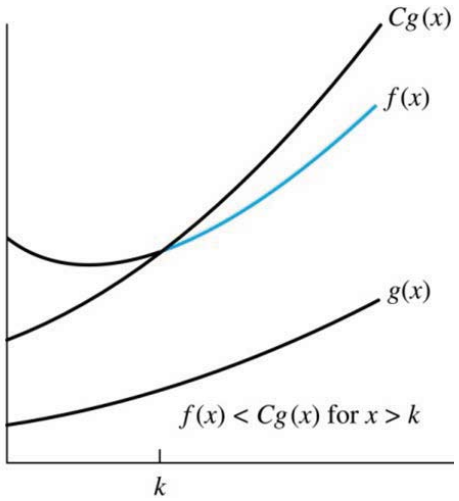
CompSci 102
Discrete Math for Computer Science



February 7, 2012

Prof. Rodger

Announcements

- Read for next time Chap. 3.1-3.3
- Homework 3 due Tuesday

- We'll finish Chapter 2 first today

Chap. 3.1 Algorithms

Definition: An *algorithm* is a finite set of precise instructions for performing a computation or for solving a problem.

Example: Describe an algorithm for finding the maximum value in a finite sequence of integers.

Solution: Perform the following steps:

1. Set the temporary maximum equal to the first integer in the sequence.
2. Compare the next integer in the sequence to the temporary maximum.
   - If it is larger than the temporary maximum, set the temporary maximum equal to this integer.
3. Repeat the previous step if there are more integers. If not, stop.
4. When the algorithm terminates, the temporary maximum is the largest integer in the sequence.

Specifying Algorithms

- Algorithms can be specified in English or in *pseudocode.*
- Pseudocode is an intermediate step between an English and coding using a programming language.
- Appendix 3 specifies pseudocode for this book (similar to Java)
- Pseudocode helps analyze the time required to solve a problem using an algorithm, independent of the actual programming language used to implement algorithm.

Properties of Algorithms

- *Input*: An algorithm has input values from a specified set.
- *Output*: From the input values, the algorithm produces the output values from a specified set. The output values are the solution.
- *Correctness*: An algorithm should produce the correct output values for each set of input values.
- *Finiteness*: An algorithm should produce the output after a finite number of steps for any input.
- *Effectiveness*: It must be possible to perform each step of the algorithm correctly and in a finite amount of time.
- *Generality*: The algorithm should work for all problems of the desired form.

Finding the Maximum Element in a Finite Sequence

- The algorithm in pseudocode:

procedure *max*(a_1, a_2, …., a_n: integers)
  max := a_1
  for *i* := 2 to *n*
    if *max* < a_i then *max* := a_i
  return *max*{*max* is the largest element}

- Does this algorithm have all the properties listed on the previous slide?

Problem

- Describe an algorithm that determines whether a function from a finite set of integers to another finite set of integers is onto.

Solution: Algorithm

Some Example Algorithm Problems

- Three classes of problems will look at in this chapter
  1. *Searching Problems*: finding the position of a particular element in a list.
  2. *Sorting problems*: putting the elements of a list into increasing order.
  3. *Optimization Problems*: determining the optimal value (maximum or minimum) of a particular quantity over all possible inputs.

Greedy Algorithms

- *Optimization problems* minimize or maximize some parameter over all possible inputs.
- Examples:
  - Finding a route between two cities with the smallest total mileage.
  - Determining how to encode messages using the fewest possible bits.
- Solved using a *greedy algorithm*, which makes the "best" choice at each step. Making the "best choice" at each step does not necessarily produce an optimal solution to the overall problem, but in many instances, it does.
- Try to prove that this approach always produces an optimal solution, or find a counterexample to show that it does not.

Greedy Algorithms: Making Change

Example: Design a greedy algorithm for making change (in U.S. money) of n cents with the following coins: quarters (25 cents), dimes (10 cents), nickels (5 cents), and pennies (1 cent) , using the least total number of coins.

Greedy Change-Making Algorithm

Solution: Greedy change-making algorithm for n cents. The algorithm works with any coin denominations $c_1, c_2, \ldots, c_r$.

procedure *change*($c_1, c_2, \ldots, c_r$: values of coins, where $c_1 > c_2 > \ldots > c_r$; n: a positive integer)
for $i := 1$ to r
  $d_i := 0$ [d_i counts the coins of denomination c_i]
  while $n \geq c_i$
    $d_i := d_i + 1$ [add a coin of denomination c_i]
    $n = n - c_i$
[d_i counts the coins c_i]

- For the example of U.S. currency, we may have quarters, dimes, nickels and pennies, with $c_1 = 25$, $c_2 = 10$, $c_3 = 5$, and $c_4 = 1$.

Proving Optimality for U.S. Coins

- Show that the change making algorithm for *U.S.* coins is optimal.

Lemma 1: If n is a positive integer, then n cents in change using quarters, dimes, nickels, and pennies, using the fewest coins possible has at most 2 dimes, 1 nickel, 4 pennies, and cannot have 2 dimes and a nickel. The total amount of change in dimes, nickels, and pennies must not exceed 24 cents.

Proof: By contradiction

Proving Optimality for U.S. Coins

Theorem: The greedy change-making algorithm for U.S. coins produces change using the fewest coins possible.

Proof: By contradiction.

Greedy Change-Making Algorithm

- Optimality depends on the denominations available.
- For U.S. coins, optimality still holds if we add half dollar coins (50 cents) and dollar coins (100 cents).
- But if we allow only quarters (25 cents), dimes (10 cents), and pennies (1 cent), the algorithm no longer produces the minimum number of coins.
  - Give an example amount that it doesn't work for.

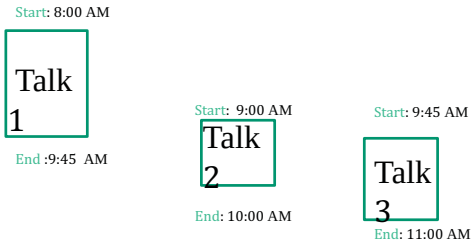
Greedy Scheduling

Example: We have a group of proposed talks with start and end times. Construct a greedy algorithm to schedule as many as possible in a lecture hall, under the following assumptions:

- When a talk starts, it continues till the end.
- No two talks can occur at the same time.
- A talk can begin at the same time that another ends.
- Once we have selected some of the talks, we cannot add a talk which is incompatible with those already selected because it overlaps at least one of these previously selected talks.
- How should we make the "best choice" at each step of the algorithm? That is, which talk do we pick ?
  - The talk that starts earliest among those compatible with already chosen talks?
  - The talk that is shortest among those already compatible?
  - The talk that ends earliest among those compatible with already chosen talks?

Greedy Scheduling

- Picking the shortest talk doesn't work.



- Can you find a counterexample here?
- But picking the one that ends soonest does work. The algorithm is specified on the next page.

Greedy Scheduling algorithm

Solution: At each step, choose the talks with the earliest ending time among the talks compatible with those selected.

procedure *schedule*($s_1 \le s_2 \le \ldots \le s_n$: start times, $e_1 \le e_2 \le \ldots \le e_n$: end times)
sort talks by finish time and reorder so that $e_1 \le e_2 \le \ldots \le e_n$
$S := \emptyset$
for $j := 1$ to n
  if talk j is compatible with S then
    S := S $\emptyset \cup$ {talk j}
return S [S is the set of talks scheduled]

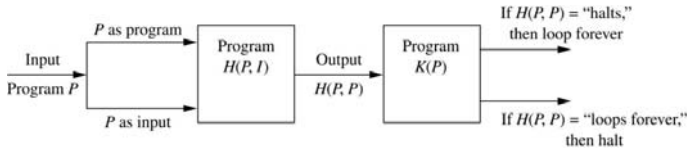
Halting Problem

Example: Can we develop a procedure that takes as input a computer program along with its input and determines whether the program will eventually halt with that input.

- **Solution**: Proof by contradiction.
- Assume that there is such a procedure and call it $H(P,I)$. The procedure $H(P,I)$ takes as input a program P and the input I to P.
  - H outputs "halt" if it is the case that P will stop when run with input I.
  - Otherwise, H outputs "loops forever."

Halting Problem

- Since a program is a string of characters, we can call $H(P,P)$. Construct a procedure $K(P)$, which works as follows.
  - If $H(P,P)$ outputs "loops forever" then $K(P)$ halts.
  - If $H(P,P)$ outputs "halt" then $K(P)$ goes into an infinite loop printing "ha" on each iteration.



Halting Problem

- Now we call K with K as input, i.e. $K(K)$.
  - If the output of $H(K,K)$ is "loops forever" then $K(K)$ halts. A Contradiction.
  - If the output of $H(K,K)$ is "halts" then $K(K)$ loops forever. A Contradiction.
- Therefore, there can not be a procedure that can decide whether or not an arbitrary program halts. The halting problem is unsolvable.

Section Summary

- Big-O Notation
- Big-O Estimates for Important Functions
- Big-Omega and Big-Theta Notation

Paul Gustav Heinrich Bachmann
(1837-1920)

Donald E. Knuth
(Born 1938)

The Growth of Functions

- Want to know how fast a function grows
- Want to understand how quickly an algorithm can solve a problem as the size of the input grows
  - compare the efficiency of two different algorithms for solving the same problem.
  - determine whether it is practical to use a particular algorithm as the input grows.

Big-*O* Notation

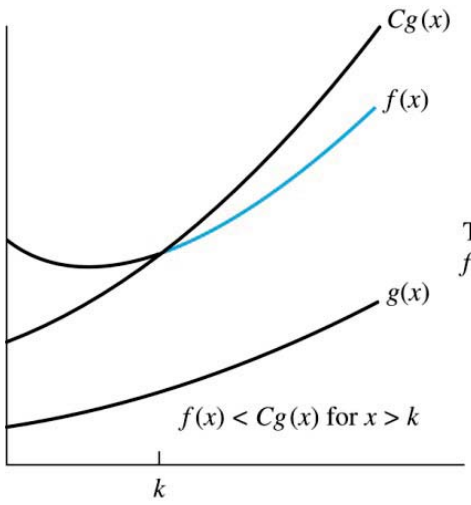
Definition: Let f and g be functions from the set of integers or the set of real numbers to the set of real numbers. We say that $f(x)$ is $O(g(x))$ if there are constants C and k such that

$$|f(x)| \leq C|g(x)|$$

whenever $x > k$.

- This is read as "$f(x)$ is big-O of $g(x)$" or "g asymptotically dominates f."

Illustration of Big-*O* Notation



f(*x*) is *O*(*g*(*x*)

The part of the graph of $f(x)$ that satisfies $f(x) < Cg(x)$ is shown in color.

Important Points about Big-*O* Notation

- If a pair C,k is found, then there are infinitely many pairs. We can always make the *k* or the *C* larger and still maintain the inequality
- $$|f(x)| \leq C|g(x)|$$
  - Any pair C' and k' where $C < C'$ and $k < k'$ is also a valid pair since $|f(x)| \leq C|g(x) \leq C'|g(x)|$ whenever $x > k' > k$.

Don't use "*f*(*x*) = *O*(*g*(*x*))" instead of "*f*(*x*) is *O*(*g*(*x*))."

- It is ok to write *f*(*x*) ∈ *O*(*g*(*x*)), because *O*(*g*(*x*)) represents the set of functions that are *O*(*g*(*x*)).

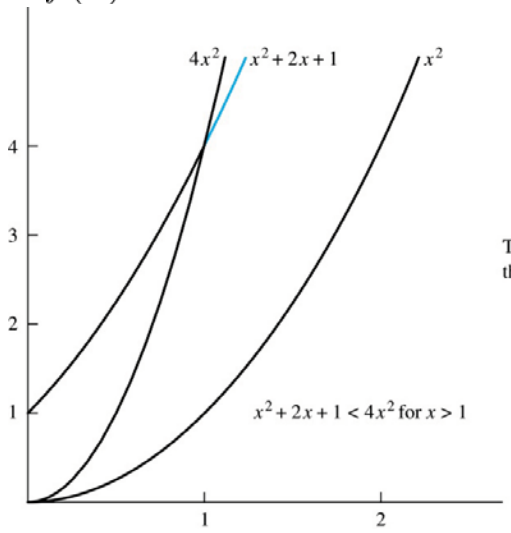
Using the Definition of Big-*O* Notation

Example: Show that $f(x) = x^2 + 2x + 1$ is $O(x^2)$.

Solution:

Illustration of Big-*O* Notation

$f(x) = x^2 + 2x + 1$ is $O(x^2)$



The part of the graph of $f(x) = x^2 + 2x + 1$ that satisfies $f(x) < 4x^2$ is shown in blue.

Big-*O* Notation

- Both $f(x) = x^2 + 2x + 1$ and $g(x) = x^2$ are such that $f(x)$ is $O(g(x))$ and $g(x)$ is $O(f(x))$. We say that the two functions are of the *same order*.
- If $f(x)$ is $O(g(x))$ and *h(x)* is larger than *g(x)* for all positive real numbers, then $f(x)$ is $O(h(x))$.
- Note that if $|f(x)| \leq C|g(x)|$ for $x > k$ and if $|h(x)| > |g(x)|$ for all x, then $|f(x)| \leq C|h(x)|$ if $x > k$. Hence, $f(x)$ is $O(h(x))$
- For many applications, the goal is to select the function $g(x)$ in *O(g(x))* as small as possible (up to multiplication by a constant, of course).

Using the Definition of Big-*O* Notation

Example: Show that $7x^2$ is $O(x^3)$.

Example: Show that n^2 is not $O(n)$.

Solution:

Big-*O* Estimates for Polynomials

Example: Let $f(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_o$ where $a_0, a_1, \ldots, a_n$ are real numbers with $a_n \neq 0$.

Then $f(x)$ is $O(x^n)$.

Proof: $|f(x)| = |a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x^1 + a_1|$ Uses triangle inequality, an exercise in Section 1.8.

$\leq |a_n| x^n + |a_{n-1}| x^{n-1} + \cdots + |a_1| x^1 + |a_1|$

Assuming $x > 1$

$= x^n (|a_n| + |a_{n-1}| /x + \cdots + |a_1| / x^{n-1} + |a_1| / x^n)$

$\leq x^n (|a_n| + |a_{n-1}| + \cdots + |a_1| + |a_1|)$

- Take $C = |a_n| + |a_{n-1}| + \cdots + |a_1| + |a_1|$ and $k = 1$. Then $f(x)$ is $O(x^n)$.
- The leading term $a_n x^n$ of a polynomial dominates its growth.

Big-*O* Estimates for some Important Functions

Example: Use big-*O* notation to estimate the sum of the first *n* positive integers.

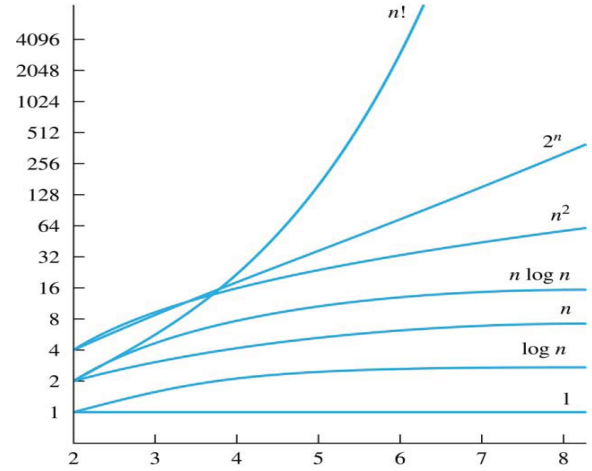
Example: Use big-*O* notation to estimate the factorial function $f(n) = n! = 1 \times 2 \times \cdots \times n$.

Continued →

Big-*O* Estimates for some Important Functions

Example: Use big-*O* notation to estimate log *n*!

Display of Growth of Functions



Note the difference in behavior of functions as *n* gets larger

Useful Big-*O* Estimates Involving Logarithms, Powers, and Exponents

- If $d > c > 1$, then
  n^c is $O(n^d)$, but n^d is not $O(n^c)$.
- If $b > 1$ and c and d are positive, then
  $(\log_b n)^c$ is $O(n^d)$, but n^d is not $O((\log_b n)^c)$.
- If $b > 1$ and d is positive, then
  n^d is $O(b^n)$, but b^n is not $O(n^d)$.
- If $c > b > 1$, then
  b^n is $O(c^n)$, but c^n is not $O(b^n)$.

Combinations of Functions

- If $f_1(x)$ is $O(g_1(x))$ and $f_2(x)$ is $O(g_2(x))$ then
  $(f_1 + f_2)(x)$ is $O(\max(|g_1(x)|, |g_2(x)|))$.
  - See next slide for proof
- If $f_1(x)$ and $f_2(x)$ are both $O(g(x))$ then
  $(f_1 + f_2)(x)$ is $O(g(x))$.
  - See text for argument
- If $f_1(x)$ is $O(g_1(x))$ and $f_2(x)$ is $O(g_2(x))$ then
  $(f_1 f_2)(x)$ is $O(g_1(x)g_2(x))$.
  - See text for argument

Combinations of Functions

- If $f_1(x)$ is $O(g_1(x))$ and $f_2(x)$ is $O(g_2(x))$ then
  $(f_1 + f_2)(x)$ is $O(\max(|g_1(x)|, |g_2(x)|))$.
  - By the definition of big-O notation, there are constants C_1, C_2, k_1, k_2 such that $|f_1(x) \le C_1|g_1(x)|$ when $x > k_1$ and $|f_2(x) \le C_2|g_2(x)|$ when $x > k_2$.
  - $|(f_1 + f_2)(x)| = |f_1(x) + f_2(x)|$
    $\le |f_1(x)| + |f_2(x)|$ by the triangle inequality $|a + b| \le |a| + |b|$
  - $|f_1(x)| + |f_2(x)| \le C_1|g_1(x)| + C_2|g_2(x)|$
    $\le C_1|g(x)| + C_2|g(x)|$ where $g(x) = \max(|g_1(x)|, |g_2(x)|)$
    $= (C_1 + C_2)|g(x)|$
    $= C|g(x)|$ where $C = C_1 + C_2$
  - Therefore $|(f_1 + f_2)(x)| \le C|g(x)|$ whenever $x > k$, where $k = \max(k_1, k_2)$.

Ordering Functions by Order of Growth

- Put the functions below in order so that each function is big-O of the next function on the list.
- $f_1(n) = (1.5)^n$
- $f_2(n) = 8n^3 + 17n^2 + 111$
- $f_3(n) = (\log n)^2$
- $f_4(n) = 2^n$
- $f_5(n) = \log(\log n)$
- $f_6(n) = n^2(\log n)^3$
- $f_7(n) = 2^n(n^2 + 1)$
- $f_8(n) = n^3 + n(\log n)^2$
- $f_9(n) = 10000$
- $f_{10}(n) = n!$

Big-Omega Notation

Definition: Let f and g be functions from the set of integers or the set of real numbers to the set of real numbers. We say that $f(x)$ is $\Omega(g(x))$ if there are constants C and k such that $|f(x)| \ge C|g(x)|$ when $x > k$.

- We say that "$f(x)$ is big-Omega of $g(x)$."
- Big-O gives an upper bound on the growth of a function, while Big-Omega gives a lower bound.
- $f(x)$ is $\Omega(g(x))$ if and only if $g(x)$ is $O(f(x))$. This follows from the definitions.

Ω is the upper case version of the lower case Greek letter ω.

Big-Omega Notation

Example: Show that $f(x) = 8x^3 + 5x^2 + 7$ is $\Omega(g(x))$ where $g(x) = x^3$.

Big-Theta Notation

Θ is the upper case version of the lower case Greek letter θ.

- **Definition**: Let *f* and *g* be functions from the set of integers or the set of real numbers to the set of real numbers. The function $f(x)$ is $\Theta(g(x))$ if $f(x)$ is $O(g(x))$ and $f(x)$ is $\Omega(g(x))$
- We say that "f is big-Theta of *g*(*x*)" and also that "*f*(*x*) is of *order g*(*x*)" and also that "*f*(*x*) and *g*(*x*) are of the *same order*."
- $f(x)$ is $\Theta(g(x))$ if and only if there exists constants C_1, C_2 and *k* such that $C_1 g(x) < f(x) < C_2 g(x)$ if $x > k$. This follows from the definitions of big-*O* and big-Omega.

Big Theta Notation

Example: Show that the sum of the first *n* positive integers is $\Theta(n^2)$.

Solution: Let $f(n) = 1 + 2 + \cdots + n$.

- We have already shown that $f(n)$ is $O(n^2)$.
- To show that $f(n)$ is $\Omega(n^2)$, we need a positive constant *C* such that $f(n) > Cn^2$ for sufficiently large *n*. Summing only the terms greater than $n/2$ we obtain the inequality

$$\begin{aligned} 1 + 2 + \cdots + n &\geq \lceil n/2 \rceil + (\lceil n/2 \rceil + 1) + \cdots + n \\ &\geq \lceil n/2 \rceil + \lceil n/2 \rceil + \cdots + \lceil n/2 \rceil \\ &= (n - \lceil n/2 \rceil + 1)\lceil n/2 \rceil \\ &\geq (n/2)(n/2) = n^2/4 \end{aligned}$$

- Taking $C = ¼$, $f(n) > Cn^2$ for all positive integers *n*. Hence, $f(n)$ is $\Omega(n^2)$, and we can conclude that $f(n)$ is $\Theta(n^2)$.

Big-Theta Notation

Example: Show that $f(x) = 3x^2 + 8x \log x$ is $\Theta(x^2)$.

Solution:

Big-Theta Notation

- When $f(x)$ is $\Theta(g(x))$ it must also be the case that $g(x)$ is $\Theta(g(x))$.
- Note that $f(x)$ is $\Theta(g(x))$ if and only if it is the case that $f(x)$ is $O(g(x))$ and $g(x)$ is $O(f(x))$
-

Big-Theta Estimates for Polynomials

Theorem: Let $f(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_o$ where $a_0, a_1, \ldots, a_n$ are real numbers with $a_n \neq 0$.
Then $f(x)$ is of order x^n (or $\Theta(x^n)$).
(The proof is an exercise.)

Example:

The polynomial $f(x) = 8x^5 + 5x^2 + 10$ is order of

The polynomial $f(x) = 8x^{199} + 7x^{100} + x^{99} + 5x^2 + 25$ is order of